

Using SpF to Achieve Petascale for Legacy Pseudospectral Applications

Dynamics of Planetary and Stellar Interiors Conference
July 28-31, 2014 — La Jolla, CA

Thomas Clune
Weiyuan Song

NASA Goddard Space Flight Center

July 29, 2014

T. Clune

S

PSIC

1/31

View metadata, citation and similar papers at core.ac.uk

provided by NASA Technical Reports Server

brought to you by  CORE

Background/Motivation

NASA HEC supports at least 5 pseudospectral applications:

Spherical Geometry

- DYNAMO
- MoSST
- ASH

Cartesian Geometry

- HPS
- DDSCAT

Background/Motivation

NASA HEC supports at least 5 pseudospectral applications:

Spherical Geometry

- DYNAMO
- MoSST
- ASH

Cartesian Geometry

- HPS
- DDSCAT

All except ASH use a 1D decomposition:

Background/Motivation

NASA HEC supports at least 5 pseudospectral applications:

Spherical Geometry

- DYNAMO
- MoSST
- ASH

Cartesian Geometry

- HPS
- DDSCAT

All except ASH use a 1D decomposition:

- limits scalability/performance
- constrains **grid resolution**

Background/Motivation

NASA HEC supports at least 5 pseudospectral applications:

Spherical Geometry

- DYNAMO
- MoSST
- ASH

Cartesian Geometry

- HPS
- DDSCAT

All except ASH use a 1D decomposition:

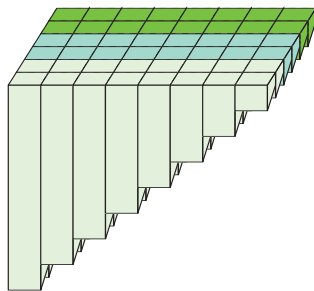
- limits scalability/performance
- constrains grid resolution



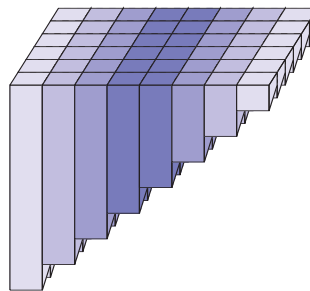
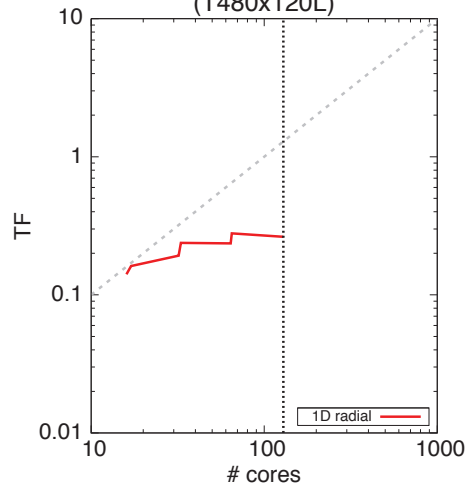
(Mostly) my fault!

Consequences of 1D decomposition

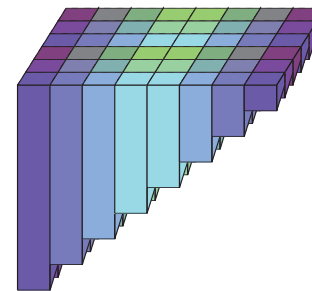
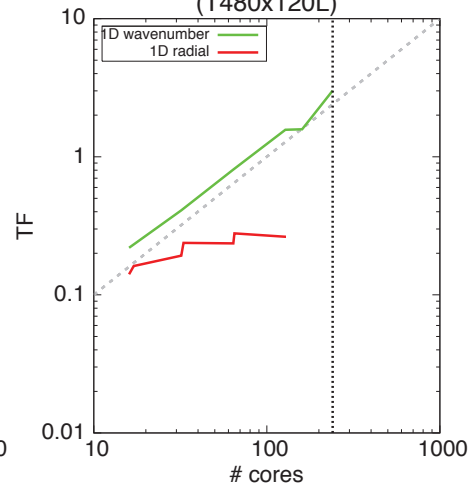
Scaling Legendre Transforms



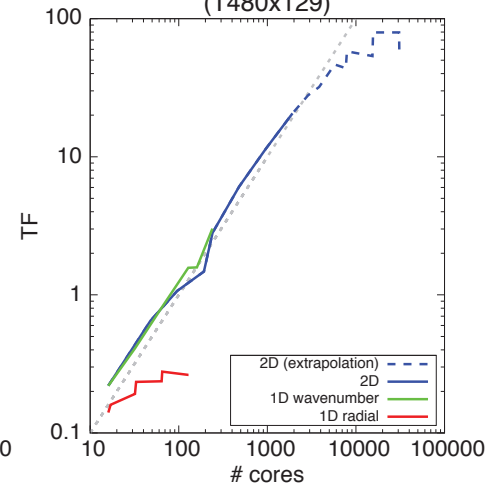
1D Radial
(T480x120L)



1D Wavenumber
(T480x120L)



2D Extrapolation
(T480x129)



Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

- Alternate between *local* computation and all-to-all communication

Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

- Alternate between *local* computation and all-to-all communication
- Complicated data structures (harmonic truncation)

Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

- Alternate between *local* computation and all-to-all communication
- Complicated data structures (harmonic truncation)
- Nontrivial load-balance

Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

- Alternate between *local* computation and all-to-all communication
- Complicated data structures (harmonic truncation)
- Nontrivial load-balance
- Most numerical calculations can be done with vendor-optimized libraries

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- **Support multiple geometries (sphere, box, ...?)**
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- **Enable users to focus on science**

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- **Manage: domain decomposition, transpose, and I/O operations**
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- **Enable users to focus on science**

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- **Leverage optimized numerical libraries**
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- **Enable users to focus on science**

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- **Support async communication, hybrid-parallelism and HW accelerators**
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- **Enable users to focus on *science***

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- **Enable *decomposition independent* formulation of applications**
- Allow user extensions/refinements (OO)
- **Enable users to focus on *science***

Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- **Allow user extensions/refinements (OO)**
- **Enable users to focus on science**

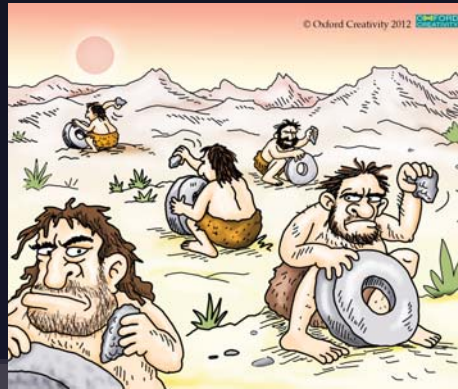
Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

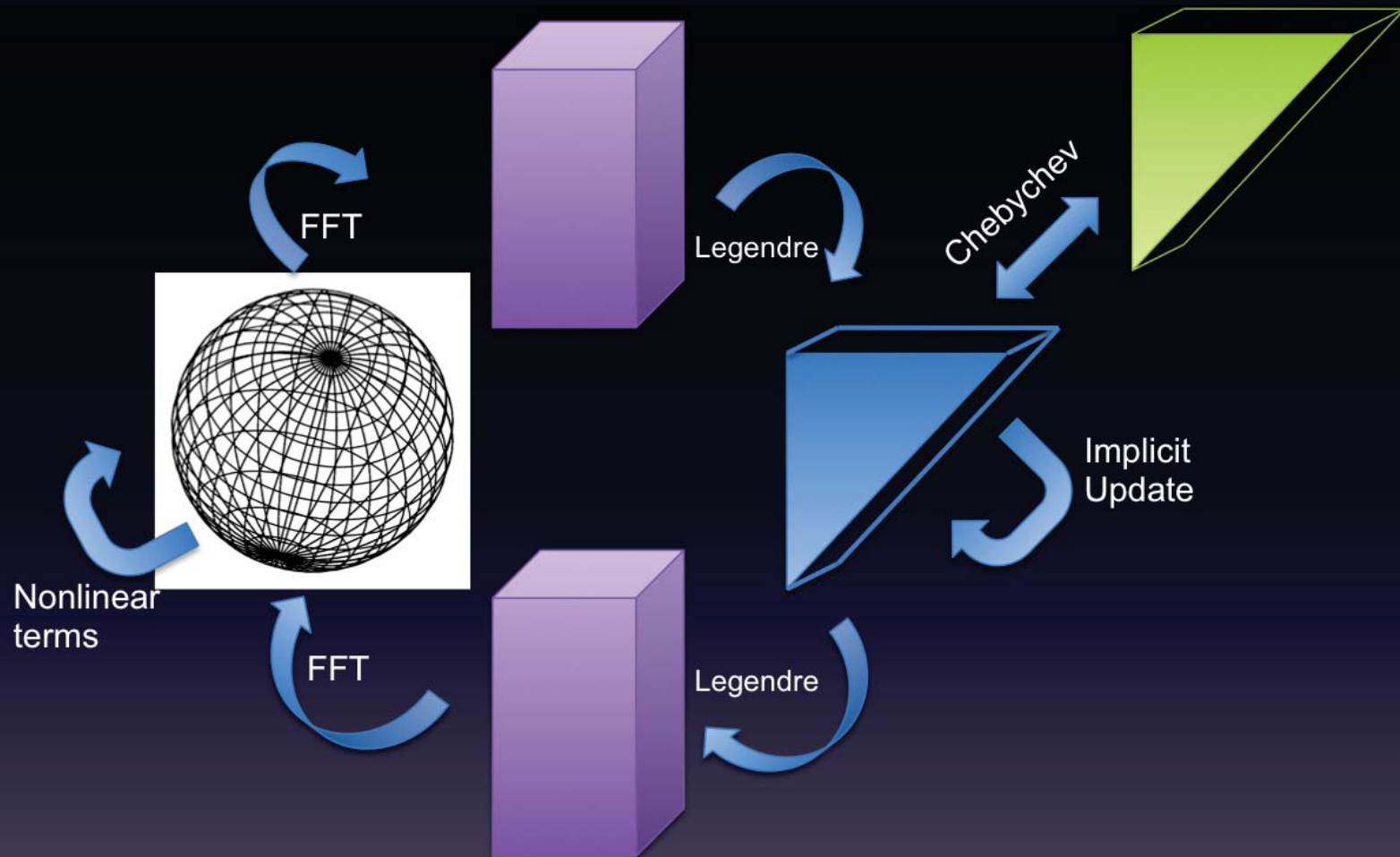
- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- **Enable users to focus on *science***

Benefits of adopting SpF

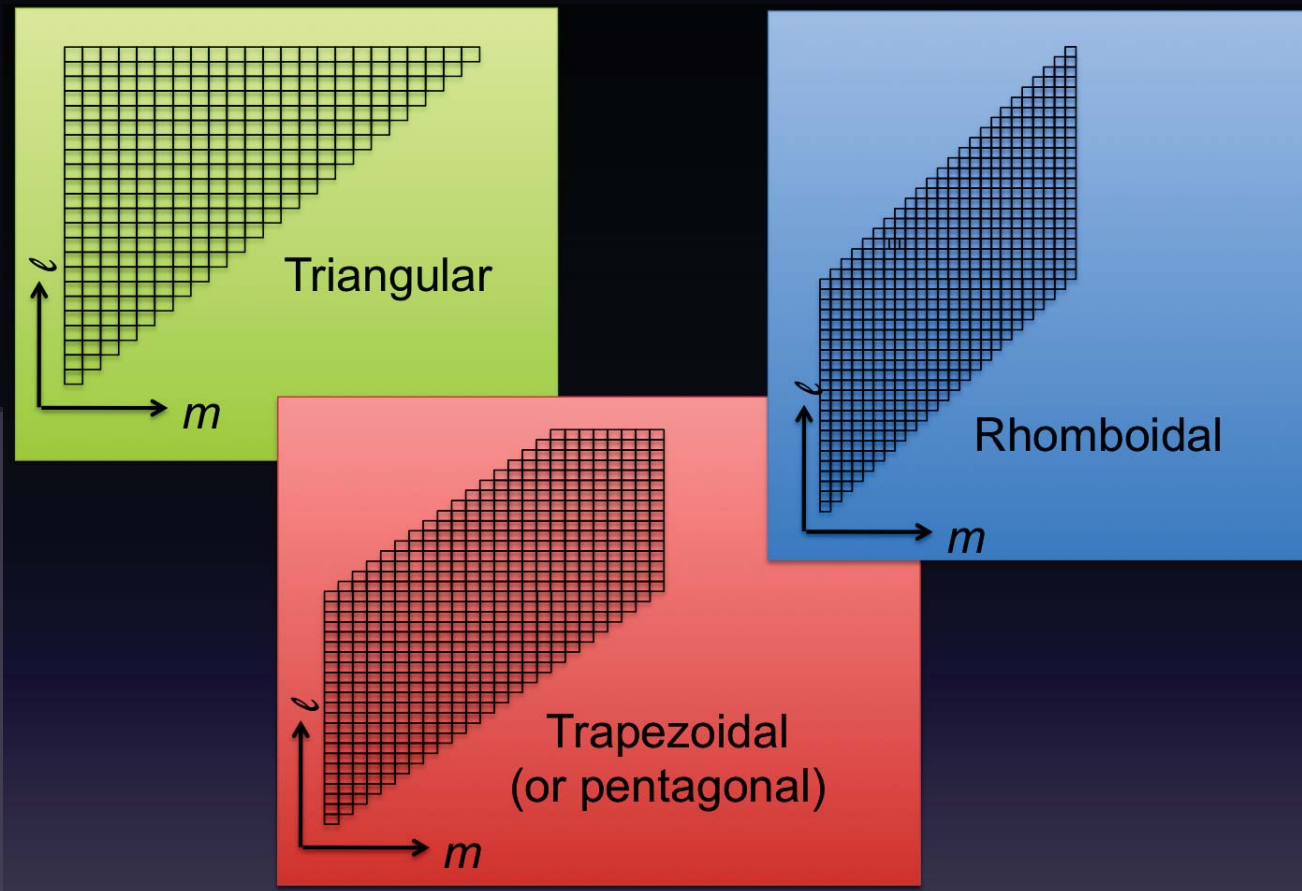


- Less duplication of effort
 - Parallel “transforms” — Legendre, LU Decomposition, etc.
 - Tedious/fragile transpose implementations
- Reduced effort to exploit new architectures/accelerators
- Readily adopt/share performance innovations within the community

Challenges and Complications



Challenges and Complications



SpF: The Secret Sauce

Each transform can be expressed as a union of independent, atomic (indivisible) numerical 'kernels' $\{K_1, K_2, \dots, K_n\}$.

SpF: The Secret Sauce

Each transform can be expressed as a union of independent, atomic (indivisible) numerical ‘kernels’ $\{K_1, K_2, \dots, K_n\}$.

These provide a natural partition of the computational domain:

$$(X, d_x) = \left((\tilde{X}_q^1, d_x^1 \otimes q^1) \oplus (\tilde{X}_q^2, d_x^2 \otimes q^2) \oplus \dots \oplus (\tilde{X}_q^n, d_x^n \otimes q^n) \right)$$

$$(Y, d_y) = \left(\tilde{Y}_q^1, d_y^1 \otimes q^1 \right) \oplus \left(\tilde{Y}_q^2, d_y^2 \otimes q^2 \right) \oplus \dots \oplus \left(\tilde{Y}_q^n, d_y^n \otimes q^n \right)$$

$$Y = F(X) \implies \tilde{Y}_q^i = K_i(\tilde{X}_q^i), i = 1, 2, \dots, n$$

SpF: The Secret Sauce

Each transform can be expressed as a union of independent, atomic (indivisible) numerical ‘kernels’ $\{K_1, K_2, \dots, K_n\}$.

These provide a natural partition of the computational domain:

$$(X, d_x) = \left((\tilde{X}_q^1, d_x^1 \otimes q^1) \oplus (\tilde{X}_q^2, d_x^2 \otimes q^2) \oplus \dots \oplus (\tilde{X}_q^n, d_x^n \otimes q^n) \right)$$

$$(Y, d_y) = \left(\tilde{Y}_q^1, d_y^1 \otimes q^1 \right) \oplus \left(\tilde{Y}_q^2, d_y^2 \otimes q^2 \right) \oplus \dots \oplus \left(\tilde{Y}_q^n, d_y^n \otimes q^n \right)$$

$$Y = F(X) \implies \tilde{Y}_q^i = K_i(\tilde{X}_q^i), i = 1, 2, \dots, n$$

Legendre

$$d_x^m = \{\ell\}_{\ell=m}^{\ell_m^{\max}} \otimes \{m\}$$

$$d_y^m = \{\theta_i\}_{i=1}^{n_i} \otimes \{m\}$$

$$q = q^m = \{r_k\}_{k=1}^{k=n_k} \otimes \{v_r, v_\theta, \dots\}$$

FFT

$$d_x = \{m\}_{m=0}^{m^{\max}}$$

$$d_y = \{\phi_j\}_{j=1}^{n_j}$$

$$q = \{\theta_i\}_{i=1}^{i=n_i} \otimes \{r_k\}_{k=1}^{n_k} \otimes \{v_r, v_\theta, \dots\}$$

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs
- Field - Local array and associated IndexSpace
 - Lingua franca within the framework
 - Input/output for kernel

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs
- Field - Local array and associated IndexSpace
 - Lingua franca within the framework
 - Input/output for kernel
- FieldList - Local collection of Field objects
 - In/out for permutation

SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs
- Field - Local array and associated IndexSpace
 - Lingua franca within the framework
 - Input/output for kernel
- FieldList - Local collection of Field objects
 - In/out for permutation
- LinearSolver - Implicit updates

SpF: Key Software Abstractions

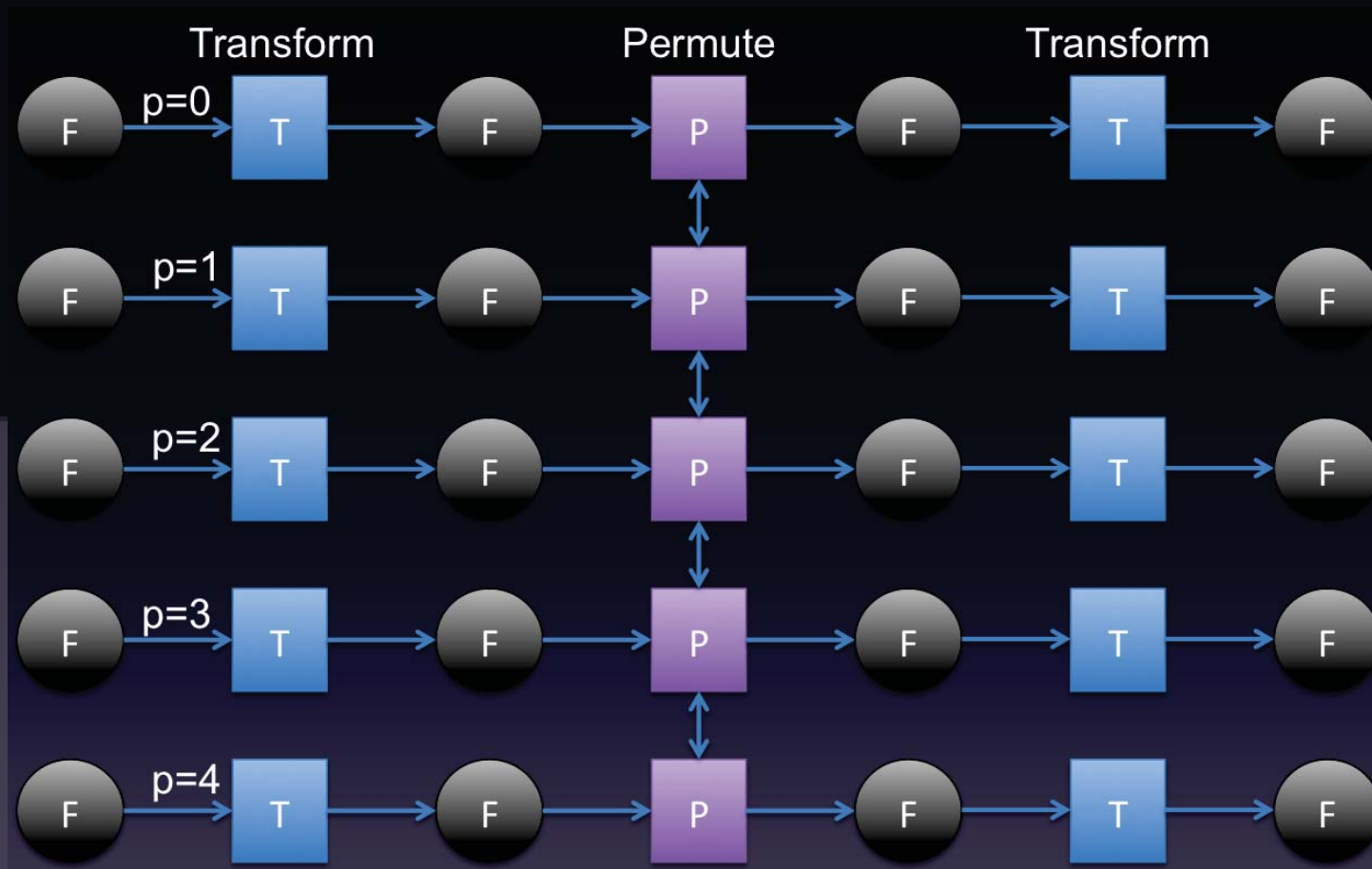
- Kernel - Indivisible unit of algorithm
 - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs
- Field - Local array and associated IndexSpace
 - Lingua franca within the framework
 - Input/output for kernel
- FieldList - Local collection of Field objects
 - In/out for permutation
- LinearSolver - Implicit updates
- Integrator - time integration (CN, AB, ...)

SpF: Implementation details

- Object-oriented design (ala Fortran 2003)
 - Applications built by *extending* SpF abstractions
 - User-extensions that can be shared by community
- Aggressive use of test-driven development (TDD) & pFUnit
 - More than 300 unit tests
 - Runs on at least 3 compilers (Intel, GNU, NAG)
- Demonstrated with multi-layer shallow water
- Not quite ready for distribution
 - Open source release planned (tedious paperwork)
 - Project-level release could be expedited



How SpF sees an application



IndexSpace - Cartesian Example

```
1 use SpF_mod
2 class (IndexSpace) :: cartesian
3 type (RangeAxis) :: xAxis, yAxis, zAxis
4
5 xAxis = RangeAxis('x', nx)
6 yAxis = RangeAxis('y', ny)
7 zAxis = RangeAxis('z', nz)
8
9 allocate(cartesian, source= xAxis*yAxis*zAxis)
```

IndexSpace - Cartesian Bundle

```
1 use Spf_mod
2 class (IndexSpace) :: cartesianBundle
3 type (RangeAxis) :: xAxis, yAxis, zAxis
4 type (StringAxis) :: qtys
5
6 xAxis = RangeAxis('x', nx)
7 yAxis = RangeAxis('y', ny)
8 zAxis = RangeAxis('z', nz)
9 qtys = StringAxis('qty', ['W', 'Z', 'S', 'P'])
10
11 allocate(cartesianBundle, source= &
12     &  xAxis*yAxis*zAxis*qtys)
```

IndexSpace - Triangular Truncation

```
1 use SpF_mod
2 class (IndexSpace) :: tDomain
3 class (OuterProductSpace) :: modeAxis
4 type (RangeAxis) :: rAxis
5
6 modeAxis = RangeAxis('m', 0, 0) * RangeAxis('ell', 0, Lmax)
7 Allocate(tDomain, source=mode)
8
9 do m = 1, mMax
10     modeAxis = RangeAxis('m', m, m) * RangeAxis('ell', m, Lmax)
11     allocate(tDomain, source= tDomain + modeAxis)
12 end do
13
14 allocate(tDomain, source= RangeAxis('r', nn)*tDomain)
```

Automating the transpose

First we translate the index space into a labelled table:

ℓ	m	r	f
0	0	1	'S'
1	0	1	'S'
\vdots			
10	7	10	'Z'
\vdots			

ℓ	m	r	f
7	2	3	'W'
7	2	4	'W'
\vdots			
0	12	2	'P'
\vdots			

Automating the transpose

Then we append process and offset metadata:

ℓ	m	r	f	PE	δ
0	0	1	'S'	0	0
1	0	1	'S'	0	1
\vdots				\vdots	
10	7	10	'Z'	8	15
\vdots				\vdots	

ℓ	m	r	f	PE	δ
7	2	3	'W'	0	0
7	2	4	'W'	0	1
\vdots				\vdots	
0	12	2	'P'	8	15
\vdots				\vdots	

Automating the transpose

Then we append process and offset metadata:

ℓ	m	r	f	PE	δ
0	0	1	'S'	0	0
1	0	1	'S'	0	1
\vdots				\vdots	
10	7	10	'Z'	8	15
\vdots				\vdots	

ℓ	m	r	f	PE	δ
7	2	3	'W'	0	0
7	2	4	'W'	0	1
\vdots				\vdots	
0	12	2	'P'	8	15
\vdots				\vdots	

Then we “co-sort” the tables to find source/destination for each element

ℓ	m	r	f	p_{src}	δ_{src}	p_{dest}	δ_{dest}
0	0	1	'S'	0	0	3	7
1	0	1	'S'	0	1	10	1
\vdots				\vdots		\vdots	
10	7	10	'Z'	8	15	2	9
\vdots				\vdots		\vdots	

Automating the transpose

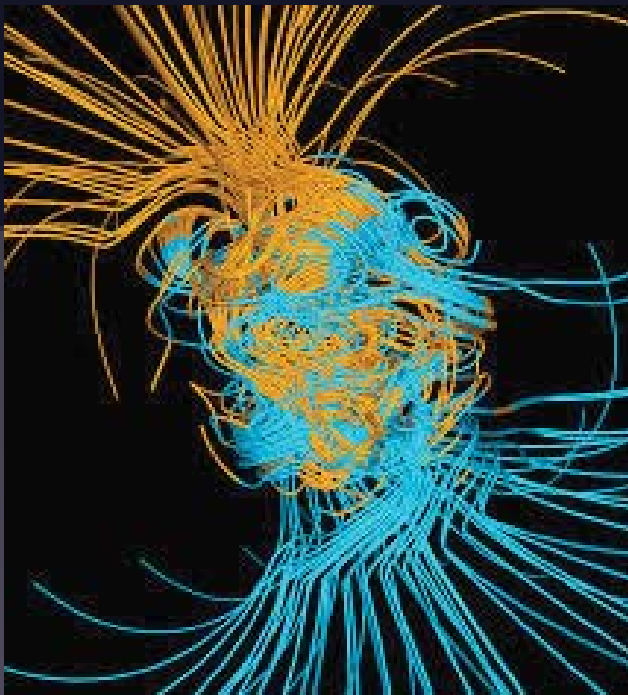
For a 2-phase (nested) transposition, we append the rank for each phase

ℓ	m	r	f	PE_0	PE_1	δ
0	0	1	'S'	0	0	0
1	0	1	'S'	0	0	1
\vdots				\vdots		
10	7	10	'Z'	8	2	15
\vdots				\vdots		

ℓ	m	r	f	PE_0	PE_1	δ
7	2	3	'W'	0	0	0
7	2	4	'W'	0	0	1
\vdots				\vdots		
0	12	2	'P'	8	2	15
\vdots				\vdots		

DYNAMO

Author: Gary Glatzmaier

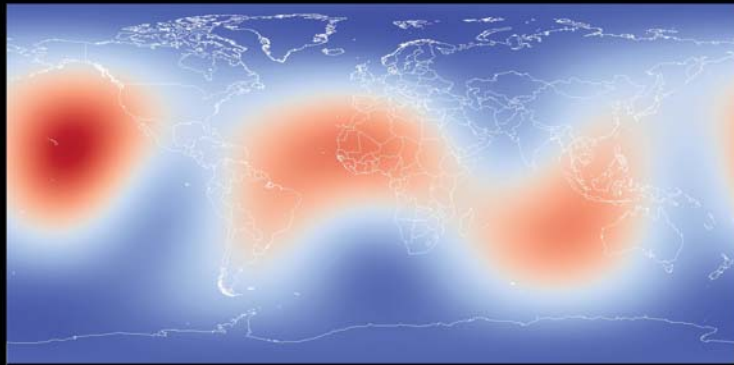


Primary configuration

- Azimuthal wavenumbers distributed over PEs
- Constraint $N_p \leq N_m$
- Supports variant spectral truncations and variant hyperviscosity terms

MoSST

Author: Weijia Kuang



Primary configuration

- One dimensional Distribution over PEs at all stages
- Constraint: $N_p \leq N_m$
- Constraint: $N_p \leq N_r$
- Constraint: All Spherical transforms (Legendre and FFT) are in the same process

Adopting SpF - general strategy

- 1 **Establish regression tests and data for baseline.**
 - Invest in achieving *strong reproducibility*
 - Turn off optimization and turn on debugging flags
- 2 Proceed with incremental changes that preserve results
- 3 Commit to repository after each success.
- 4 Minor roundoff issues may be encountered
 - Isolate cause, then update baseline regression data
 - Bracket change in repository

Adopting SpF - copy to/from legacy data structures

- 1 Declare a FieldList object
- 2 Create a procedure that copies an array into a Field
- 3 For each contiguous array
 - 1 Define corresponding IndexSpace domain object
 - 2 Call append() method on FieldList
 - 3 insert call to copy procedure just prior to use

Adopting SpF - Kernel Factory

- 1 Create a new module:
 - 1 Define a derived type that *extends* KernelFactory
 - 2 Implement methods that compute Kernel IndexSpace (I/O)
 - 3 Define a derived type that *extends* Kernel
 - 4 Implement apply() method that wraps actual computation
- 2 Declare and initialize in main code:
 - 1 new Factory defined above
 - 2 Distributor, Permutor
 - 3 TaskList, and 2 FieldLists (in and out)
 - 4 Build task list, and field lists using distributor and factory
 - 5 Build permutor object connecting previous transform to new
- 3 Use in main loop:
 - 1 Insert call to apply() method of TaskList object

Adoption status

MoSST

- Now uses SpF permutations

DYNAMO

- SpF conversion completed for
 - Legendre transforms
 - Quadratic convolution
 - Stream to vector (i.e. $\{W, Z, \dots\} \longrightarrow \{v_r, v_\theta, \dots\}$)
 - Permutations (including to/from legacy layout)
- Took ≈ 1 week for expert (me)
 - Lots of ugly shortcuts
- Issues encountered with implicit update step
 - Could “cheat”
 - Will use experience to instead improve framework

Example - top declaration

```
1 type (SimpleMpiDistributor) :: d
2 type (FieldList) :: leg_in, leg_out, NL_in, NL_out
3 type (LegendreFactory) :: legFactory
4 type (NL_ConvolutionFactory) :: NL_Factory
5 type (PartitionedAlgorithm) :: legTasks, NL_tasks
6 type (SimpleMpIPermutor) :: perm
7 class (IndexSpace) :: initialDomain
8
9 d = SimpleMpiDistributor(MPI_communicator)
10 legFactory = LegendreFactory(mMax=1023)
11 NL_Factory = NL_ConvolutionFactory(ni, nk)
12 initialDomain = ...
```

Example - initialization

```
1 legTasks = d% distribute(legFactory, initialDomain)
2 leg_in = FieldList(legTasks, 'in')
3 leg_out = FieldList(legTasks, 'out')
4
5 NL_tasks = d% distribute(NL_Factory, leg_in)
6 NL_in = FieldList(NL_tasks, 'in')
7 NL_out = FieldList(NL_tasks, 'out')
8
9 perm = SimpleMpiPermutor(MPI_communicator, leg_out, NL_in)
```

Example - execute

```
1 ...  
2 call legTasks%apply(leg_in, leg_out)  
3 call perm%permute(leg_out, NL_in)  
4 call NL_tasks%apply(NL_in, NL_out)  
5 ...
```

Variations

Variations

```
1 ! Alternate load balancing strategy
2 ! type (SimpleMpiDistributor) :: d
3 type (RoundRobinDistributor) :: d
```

Variations

```
1 ! Alternate load balancing strategy
2 ! type (SimpleMpiDistributor) :: d
3 type (RoundRobinDistributor) :: d
```

```
5 ! Alternative permutation strategy
6 ! type (SimpleMpiPermutor) :: perm
7 type (SomeOtherPermutor) :: perm
```

Variations

```
1 ! Alternate load balancing strategy
2 ! type (SimpleMpiDistributor) :: d
3 type (RoundRobinDistributor) :: d
```

```
5 ! Alternative permutation strategy
6 ! type (SimpleMpiPermutor) :: perm
7 type (SomeOtherPermutor) :: perm
```

```
9 ! Alternative Legendre implementation
10 ! type (LegendreFactory) :: legFactory
11 type (AltLegFactory) :: legFactory
```

Next steps

- Finish ports of DYNAMO, MoSST, HPS, DDSCAT
- Improve framework
 - Generalize/optimize Permutor classes
 - Allow for multiple sources
 - Allow for “subsetting”
 - Implement multiphase transpose (ala Nick Featherstone)
 - Extend/improve kernels
 - Better mechanism for defining offsets
 - Allow for multiple sources/destinations
 - Allow for “fat” kernels that do internal communication (e.g. implicit treatment of coriolis)
- Release SpF as open source

Credits

- NASA High End Computing program for supporting this work
- Gary Glatmaier - for providing DYNAMO as an interesting challenge

Questions?